Association for Information Systems
AIS Electronic Library (AISeL)

MCIS 2011 Proceedings

Mediterranean Conference on Information Systems (MCIS)

2011

UNDERSTANDING DOCUMENTATION'S IMPORTANCE – TREAT OTHERS AS YOU WOULD WANT TO BE TREATED

Aharon Yadin
Academic College of Emek Yezreel, aharony@yvc.ac.il

UNDERSTANDING DOCUMENTATION'S IMPORTANCE – TREAT OTHERS AS YOU WOULD WANT TO BE TREATED

Yadin, Aharon, The Max Stern Academic College of Emek Yezreel, Israel,
aharony@yvc.ac.il

Abstract

This research for using team based role play for enhancing understanding was performed within a Software Engineering workshop and is based on a previous study. The main research objective was to demonstrate the importance of documentation as part of the Software Development Life Cycle (SDLC). The methodology for achieving the objective was based mainly on a unique course structure. As part of the workshop, students are required to develop a real project. For stressing the importance of software engineering on the whole development process, the workshop consists of three different stages: analysis and design, development and testing. The students have to complete three assignments which correspond to the three stages. However, contrary to the ordinary structures for such courses, in which the three stages are performed by the same team, in this workshop each project's, stages were performed by three different teams. As a result, each team had to continue the work performed by the previous team, which created inter-dependency between the teams. Furthermore, the teams' performance and grades depend on each other and especially on the previous assignment's proper documentation. The research methodology was based on two similar questionnaires for assessing the students' perception regarding documentation. The first questionnaire was given during the first lecture and the second questionnaire after the last lecture. The difference between the two questionnaires for each student reveals the change in perception achieved during the workshop. The research, which was performed during two years, demonstrated that employing this type of team based role play during the SDLC enhanced the students' understanding regarding the importance of documentation, mainly by exposing them to difficulties and consequences caused by its absence.

Keywords: Team Based Role Play, Software Engineering, Software Documentation

1 INTRODUCTION

The term software engineering was first coined at a NATO (North Atlantic Treaty Organization) conference forty years ago (Naur and Randell, 1968). The conference was intended to stimulate discussions on the issue of software development and especially on developing correct, testable and understandable computer programs. In the last four decades, software engineering has grown and is currently accepted as a well known and proven learning discipline, an important part of the Computer Science (CS) and Information Systems (IS) curricula. Like many other products, Information Systems require follow-up maintenance for correction of newly discovered bugs. However, unlike other products and due to the important, sometimes critical role, Information Systems play in the organization, maintenance is also used for enhancing the functional capabilities to the system. For large software development projects, which usually require significant human resources for maintenance, adopting proper software engineering methodologies may lower the costs associated with these maintenance activities. Thus, developing maintainable software plays a critical function in the software engineering process, not unlike the role of software development itself. For that reason, in addition to the technical issues related to the whole Software Development Life Cycle (SDLC), the Software Engineering workshop stresses the importance of software maintainability and especially the significance of documentation in the process.

To address the students' difficulties regarding non-technical knowledge such as critical thinking, interpersonal and team-based skills (Davis, Thomas and Kazlauskas, 2006; Venable, 2008) the software engineering workshop structure is based on team activities. Furthermore, to raise the students' awareness of the importance of documentation and maintainability, the workshop employs an incremental life cycle structure involving each team in three activities: design with a special emphasis on documentation, development, and testing. However, unlike the ordinary software development life cycle, the workshop structure employs a team-based role-play (Yadin and Lavy, 2008). This means that the design, development and testing of the project are swapped among three teams. Initially all the teams are given the same project and they have to design the system. Due to the very generic definition of the project, and the fact that each team has to define its project's specifications, the designs are very different. The second phase is developing the system according to the design specifications. However, unlike the usual structure, in which each team develops the system they designed, here each team develops a system that was designed by a different team. The third phase consists of defining the test specifications and performing the tests. Once again, each team tests a system designed by one team and developed by another. When proceeding to the next phase of the Software Development Life Cycle, the team is required to ignore their prior knowledge or ideas and to concentrate only on the system as it has been designed (or developed) by the previous team that worked on this project.

This is a follow-on (Yadin and Lavy, 2008) research and its main objective was to further examine the effect of employing this workshop structure that uses role-based and team-based peer review on the students' learning process with special emphasis on their perceptions regarding documentation importance. This paper describes the workshop structure and the results obtained.

2 THEORETICAL BACKGROUND

Usually, software development is a shared task that employs individual developers, or teams that are working collaboratively (Cheng, Hupper, Ross, and Patterson, 2003). The issue of software readability and maintainability was addressed by many software engineering practitioners in attempts to improve existing supporting tools and methodologies. One of the important practices required for future maintenance is documentation, used to describe the required software and its performance (de Souza, Anquetil and de Oliveira, 2007; Das, Lutters and Seaman, 2007). Without proper documentation about the system and its processes, future maintenance will be extremely difficult. Unfortunately, poor

documentation is still a major contributor to software quality degradation and the increase in its maintenance costs (Kajko-Mattsson, 2008). In spite of this and the continuous efforts of educators, many students still fail to recognize the importance of maintainability (Burge, 2007).

Many practices, methodologies and tools are combined together in software engineering and one of the initial practices is software documentation. However, even with these widely available methodologies, many systems are still developed and released without proper documentation (Coleman, Ash, Lowther and Oman, 1994; Broy, Deissenboeck and Pizka, 2006). In trying to cope with the lack of documentation, several methodologies have been developed to address this unsolved problem (Clements, 2005). The Agile Manifesto, for example, puts more emphasis on "Working software over comprehensive documentation" (Beck , 2001). This methodology is about change and concentrates on fast delivery of useful software by close collaboration between developers and users. This collaboration is used to avoid the documentation issue.

The term software documentation usually refers to two types of documentation: (1) documenting the software requirements and (2) documenting the software to be developed, or which was developed. Although, the importance of documentation (during the design phase or after project completion) was stressed by practitioners and educators, many projects are still released without proper maintenance documentation.

2.1 Software Documentation

Software maintenance is difficult and expensive because the software is intangible and complex. A prerequisite for maintaining software is the understanding of the written code and the control flow between its components. Documentation plays a vital role in enhancing this understanding, especially for emerging software development technologies (Chua, Purao, and Storey, 2006).

Usually, maintenance refers to various activities that are performed after the product was delivered (Keirnan, Anschuetz and Rosenbaum, 2002). Software maintenance is no different and is performed after the development was completed. However, when compared to other types of maintenance (for example, equipment repairs), software maintenance is sometimes very different. In most engineering disciplines maintenance is used in order to fix a problem (Canfora, and Cimitile, 2000) so the product will keep on working as intended. This is similar to bug-fixing in software maintenance. However, due to the ever changing operational business environment, the software that supports it has to be changed as well. This means that unlike other types of engineering maintenance, software maintenance is sometimes required to enhance the system and provide additional new functionality. These enhancements are performed as part of the software maintenance and are unique to software as defined by Lehman's Laws of Software Evolution (Lehman, 1980; Lehman, 1984). Furthermore, software is constantly being modified to support the new hardware equipment and to integrate into new application environments.

The introduction of the SDLC influenced the importance of software documentation and especially led to the understanding that documentation activities should be initialized during software development and not only after completion or delivery. Furthermore, some researchers stress that starting the documentation activities after completing development leads to an unnecessarily more complex task (Schneidewind, 1987; Osborne and Chikofsky, 1990).

2.2 Students' Perceptions Regarding Documentation

Following the introduction of the SDLC there has been a great deal of improvement in software development. Many new techniques, methodologies and tools have been developed in order to advance the various stages of the development processes. However, due to its reactive nature, software maintenance lags behind. Furthermore, systemic approaches to software maintenance are inherently problematic (Dias, Anquetil and de Oliveira, 2003; Lientz, 1983). Attempts to delay the required software maintenance activities, either error corrections or capability enhancements, are impossible

and sometimes extremely costly. As stated by Lehman (Lehman's Second Law of Software Evolution (Lehman, 1980)), software maintenance is an unsystematic process which deteriorates the software's architecture. This deterioration is due in part to missing knowledge/documentation required for maintenance (Trienekens, Kusters et al, 2001). In addition, any changes introduced as part of the maintenance deteriorate the system architecture further, causing future maintenance to be even more complicated (Grubb and Takang, 2003).

As stated by several researchers (Karahasanović and Thomas, 2007; Zou and Kontogiannis, 2002), software maintenance uses significant costs during the SDLC. Many researchers agree that it requires more than 50% of the overall resources (Coleman, Ash, Lowther and Oman, 1994; Holgeid, Krogstie and Sjøberg, 2000; Lehman and Belady, 1985; Lenic, Zorman, Povalej and Kokol, 2004; Nosek and Prashant, 1990), so proper training of students both on the role of maintenance and the importance of documentation in the software development process is vital. During their first and second years of study, students learn and understand these issues. However, in spite of this fact software documentation continues to be insufficient. More disturbing is the fact that students do not assimilate the need for, and importance of, proper documentation for future maintenance and the overall success of the software project (Burge, 2007; Broy, Deissenboeck and Pizka, 2006).

3 THE STUDY

3.1 About The Study Participants

The present study was performed over a period of two years. This is a follow-on study (Yadin and Lavy, 2008), which was extended and used in a larger setting. A total of 51 college students in their second year of study towards a BA degree in CS and IS participated. As part of the workshop the students were divided into 14 teams (nine teams of four students and five teams of three students). Students were not allowed to replace their team during the semester. The students represent all talent levels. The workshop is a mandatory course taken during the second year of study. At this stage of their studies the students have already learned software architecture and modelling, Unified Modelling Language (UML) usage, and taken an introductory IS course stressing the importance and flexibility of IS in the business environment, etc.

3.2 The Course

The general objectives of the systems engineering workshop are to introduce SDLC concepts to the students while enhancing their understanding of documentation importance to product maintainability. Since many consider software systems as one of the most complex systems produced by humans (Lenic, Zorman, Povalej and Kokol, 2004), the course is intended to help students adopt the proper software development working procedures. Special emphasis is put on documentation, as students are required to document their thoughts and ideas during the design phase which is crucial for future understanding of the software.

One of the workshop's important objectives is to prepare the students for their final project and the real-world challenges they will face after graduation. Other, more detailed, course objectives relate to: (1) practical understanding of the software development stages; (2) implementing these stages in a small project; (3) understanding the problems associated with and caused by working in teams; and (4) developing the required "soft skills" (critical thinking, teamwork, etc). To address these objectives, the workshop augments knowledge and understanding gained in current and previous courses, and is practical and team-based.

All 14 teams received and worked on an identical project which was a general description of a required system to be developed (an Internet-based electronic auction, or e-bidding system). The students had to study the currently available systems, address and assess the various alternatives, and design (and later develop and test) their solution. The workshop structure followed the SDLC and was

based on three incremental assignments, correlating to the three stages of design, development and testing. The students had four weeks for each assignment in which they worked first by themselves, and later met together, or used collaborative tools of their choice. Throughout the process they consulted their instructor (via email, the workshop website, and personal meetings). To reinforce the students' understanding regarding the importance of documentation, the teams were engaged in role-based development. This means that each project was designed, developed and tested by three different teams with a shared responsibility for their success. Each specific project was designed by one team, developed by another team and tested by a third team. Each team worked on the three stages; however, each stage belongs to a different project (Figure 1). This way, each team was involved in developing a system designed by another team while trying to understand the intentions expressed in the design. If the design documentation was not complete, the development team had to seek help from the designing team. On the other hand, this developing team had to help another team that was trying to develop the system based on their design document. The interdependence of these stages was stressed and made apparent to all teams. This workshop structure was especially designed to enhance the students' understanding regarding the importance of documentation through their own experience.

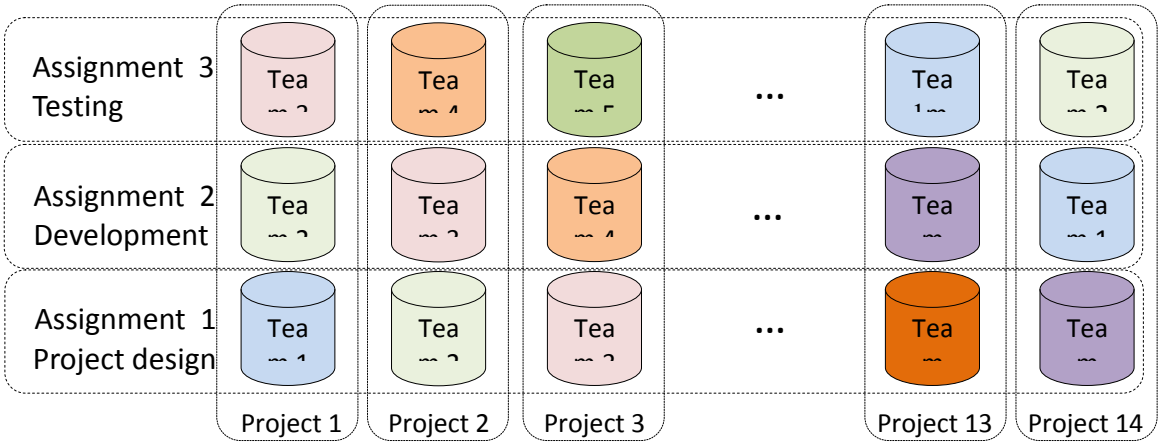

Figure 1. The workshop structure

Figure 1 depicts the workshop's structure. The long horizontal rectangles represent the three phases and assignments, while the 14 vertical columns represent the 14 projects. As can be seen, each project consists of the three assignments performed by three different teams, which are represented by different colours. Each team, on the other hand, worked on all three assignments, each one belonging to a different project. Project 1, for example was performed by teams 1 (design), 2 (development) and 3 (testing). There were two types of deliverables: (1) team assignments; and (2) a personal assignment.

3.3 Team Assignments

Following the SDLC model, the software development activities in the workshop were divided into three team-based phases and assignments:

(1) Project definition and design with a special emphasis on documentation: the first assignment started with a brief description of the project goals and the necessary system capabilities. The students were required to study some of the available e-bidding systems, documenting their functionality, and use it in defining the requirements to their specific design. In designing the system, the students had to identify at least five different users, supported by the system and for each one, define a set of use cases that are required. In addition to the sequence diagrams supporting these use cases, the students had to

define the non-functional requirements associated with these use cases. The system analysis phase (which is part of this assignment) included a high level design (system architecture and the class diagram) as well as a detailed design (activity diagram followed by a program design language definition). All these activities required a great deal of collaborative work and peer review, in which each student assessed and approved the work performed by other team members.

(2) Project development: the second phase and assignment consisted of the development of the system according to the project definition and design document, which was produced as part of the first assignment. Each team had to develop the system exactly as it was defined and designed by the other team. The developing team had to carefully follow the document they received, ignoring all their prior knowledge or ideas they might have expressed in their first design. Small code modifications were permitted, provided that the definition in the document they received was erroneous and could not be implemented. After completing the development, each team had to compile a "difference" document, outlining the changes between the developed implementation and the document as it was designed. Special emphasis was placed on the reasons behind the changes between the design document and the implementation. The students were not allowed to enhance the product to be developed, but rather to develop it according to the exact specifications outlined in the design document. An additional document which was part of this assignment was a short evaluation of the first assignment's quality as it was reflected in the implementation. The last document to be submitted as part of this assignment was a unit test plan for each of the methods developed.

(3) Project testing: the third phase and assignment consists mainly of the testing phase. The students had to implement the unit test plan as designed by the previous team. Due to time constraints the workshop addresses only part of the required project development, so the testing phase includes additional developments (a test generator and a stub) for building the infrastructure required for testing the developed software pieces. As part of this assignment the team was required, not only to test, but also to correct the mistakes that were discovered. The corrected code had to be tested once again to assure its correctness. This process is repeated until everything runs according to the specifications. As part of the assignments, the students had to provide the testing report that summarizes the problems discovered and their corrections. In addition, this assignment included a system test plan with at least 10 detailed test cases. This plan is for the quality assurance staff, so it has to be detailed and based on the system functionality as derived from the software specifications as defined in the design document (first assignment). The last part of this assignment is a quality test plan which concentrates on the non-functional attributes of the system, with a special emphasis on the metrics to be used or defined.

3.4 Personal Assignment

The personal assignment is an activity summary report, in which each student describes: (1) the work done during every stage; (2) his/her part in these activities; (3) the problems they (as a team) encountered during the project; and (4) the problems he/she encountered personally. There is also a short reflection on the workshop, as well as a one-sentence summary about the workshop's results. The last part reflects on the work distribution among the team members (100 points that the student divides between the other team members to express their relative contribution toward each of the three assignments).

4 LEARNING PROCESS EVALUATION METHODOLOGY

The workshop was highly structured and allowed for role-play between the teams. To enhance understanding and the project's success, pre-defined templates were used for all of the team-based assignments. The evaluation method included a comparison between two identical one-question questionnaires in which students were asked to rate their perception regarding the relative importance of the three planned assignments. Each student was given 100 points and was asked to divide these points between the three project stages based on his or her perception regarding the relevant importance of each stage. The first questionnaire was given during the workshop's first lecture and the

second questionnaire during the last lecture. The intention of using two identical questionnaires was to measure the workshop's influence regarding the perceived importance of development, documentation and testing on the software engineering activities of the project. In addition, the evaluation process analyzed the students' reflections on their workshop experiences. In contrast to the pre-defined templates used for the team activities, the personal reports were composed of freestyle answers. The only data provided were the points to be addressed in these reflections. This open format encouraged students to concentrate on the issues they felt were important and offered a better understanding of the students' achievements during the workshop. To examine the significance of the differences between the students' perceptions concerning the importance of the various phases of software development, before and after their engagement in the project, Paired – Sample – T Test was employed and the results are provided in the following section.

5 RESULTS AND DISCUSSION

The first questionnaire results are outlined in Figure 2 (pre-workshop). The numbers were calculated as the average between all students' answers. It is no surprise, as already discovered by a previous study (Yadin and Lavy, 2008), that CS students regard development as the most important activity (74% of the project). Testing was perceived to be of secondary importance (14%) and documenting the design phase was perceived to be the least important (12%) among the phases of the design and development of software. These results are not unexpected and support many researchers who claim that developing software is more than just knowing a programming language (Hunt and Thomas, 2000; Kernighan and Pike, 1999; McConnell, 2004).

When asked to relate to the results, the students explained the relatively low importance of documentation by referring to the fact that the methodology and the tools used (UML) provided all the necessary documentation required for the project. The results obtained in this survey were no different when compared to the results from the previous year (Yadin and Lavy, 2008), or the year before, in which no questionnaires were used and the students' perceptions were obtained from their reflections. The misconception that the generic UML charts are sufficient for development was the trigger for the workshop structure and one of its objectives was to convince students, through their own practical experience, about the importance of documentation.

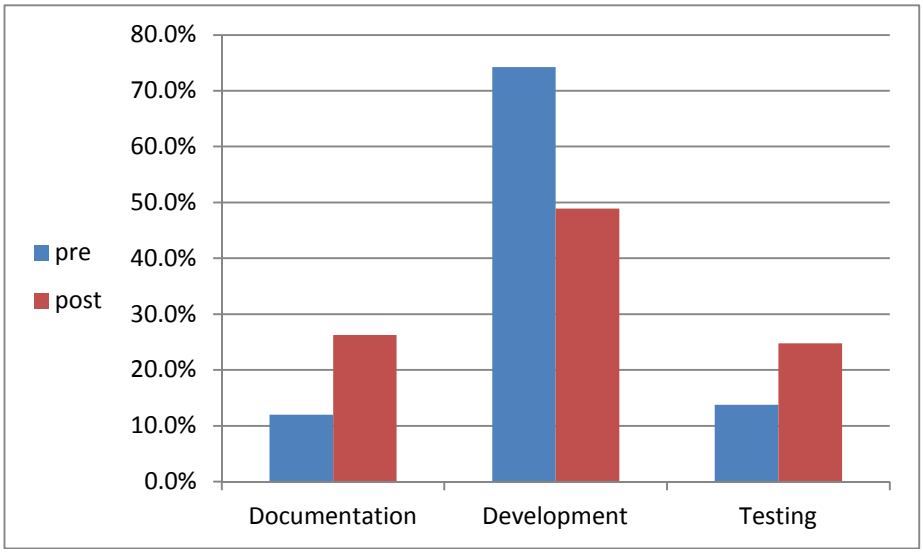

Figure 2. Relative assignment's importance

The first questionnaire (Figure 2 pre-workshop) demonstrated the students' misconception regarding the relative importance of development (74%). However, the second questionnaire revealed that the students began to realize the importance of the other phases as well as the vital role they play in determining the project's success as demonstrated by Figure 2 (post workshop).

Analyzing the differences between the two figures reveals that the students' perception changed significantly. The relative importance of documentation increased by 119%, ($t(50) = -18.08$, $p<.001$), while the relative importance of the coding stage dropped by 34% ($t(50) = 18.97$, $p<.001$). The importance of testing was increased by 80% ($t(50) = -8.30$, $p<.001$). There are many factors affecting the relative importance of the various life cycle stages and the amount of time required for each one. These factors may include, for example, the user requirements, the project type, the SDLC methodology used, the programming languages and Computer Aided Software Engineering (CASE) tools, etc. In many cases the development phase requires less than 30% of the project's estimated time. Glass (Glass, 2003) uses 20% for requirements elicitation, 20% for design, 20% for coding and 40% for testing. The requirements elicitation is not part of this workshop since the project and its requirements were predefined and the students had to decide which parts to design and implement. In addition, no real users were involved and the project's requirements and specifications were determined by the students themselves. At the end of the semester, the students still regard coding (development) as the most important component, but it is significantly (34%) less than its perceived importance at the beginning of the semester. The end of semester percentages are closer to the numbers used by researchers and practising software engineers, for example the percentages proposed by Glass (Glass, 2003).

The alteration in the students' perception regarding the relative importance of the various project components is directly linked to the workshop's structure. This became evident mainly at the role-play changing points. During the second phase of the workshop in which the students had to develop the system based on the design they received, some of the teams struggled to continue their work. Several students were trying to drop the design specifications they received from the previous team, claiming these specifications would not produce a workable solution and the project could not be developed. In all these cases and after some additional thinking this proved to be wrong. The solutions described in these design specifications provided a viable solution, but they were not properly documented, which hindered the students' understanding. After discussing the design specifications with the team responsible, the project was developed as intended, sometimes with some minor modifications. These types of misunderstanding were repeated in the third phase, in which students had to design and execute the system testing. Once again, for designing the test environment and the test scenarios, a full project understanding was required, but without the proper documentation, this understanding was extremely difficult. In addition to the extra work required due to the missing documentation, this also changed the students' perception regarding the importance of the other, non-development, "soft skills" based activities.

The significant increase in the perceived importance of testing (80%) can be explained by the fact that during the third stage the students had to design and develop the stub and the scenarios for the system to be checked. In all cases where the system did not function according to the specifications, the testing team had to correct the code and run it once again. This means that the testing team had to be familiar with the design specifications as well as with the developed code. The testing students acted as the "gate-keepers", making sure that only the fully functional system was released. Performing this task properly required some additional analytical skills to find and correct various bugs which might have been introduced during the development or the design phases. Unlike in the real-world situation, where in the case of problems, quality assurance people usually return the system to the developers, here the testing team had to fix it, which led to their higher appreciation of the testing task and its perceived elevated importance in the development process.

References

Beck, K. (2001) “The Agile Manifesto”, http://agilemanifesto.org (current February 2011).

Broy, M., F. Deissenboeck and M. Pizka (2006) “Demystifying Maintainability”, in Proceedings of the 4th Workshop on Software Quality, Shanghai, P.R.C., ACM Press.

Burge, J. (2007) “Exploiting Multiplicity to Teach Reliability and Maintainability in a Capstone Project”, 20th Conference on Software Engineering Education & Training (CSEET'07).

Canfora, G. and A. Cimitile (2000) “Software Maintenance”, http://www.compaid.com/caiInternet/ezine/maintenance-canfora.pdf (current January 2011).

Cheng, L. T., S. Hupper, S. Ross, and J. Patterson. (2003) “Jazzing up Eclipse with Collaborative Tools”, Proceedings of the 2003 OOPSLA Workshop on Eclipse Technology Exchange, Anaheim, October 2003.

Chua, C.E.H., S. Purao, and V.C. Storey (2006) “Developing Maintainable Software: The Readable Approach”, Decision Support Systems, 42(1), pp. 469-491.

Clements, P. (2005) “Comparing the SEI’s Views and Beyond Approach for Documenting Software Architectures with ANSI-IEEE 1471-2000”, Technical Note. CMU/SEI-2005-TN-017 http://www.sei.cmu.edu/pub/documents/05.reports/pdf/05tn017.pdf (current January 2011).

Coleman, D., D. Ash, B. Lowther and P. Oman (1994) “Using Metrics to Evaluate Software System Maintainability”, IEEE Computer, August 1994, pp. 44-49.

Daich, G. T. (2002) “Document Diseases and Software Malpractice”, http://www.sstc.online.org/Proceedings/2003/PDFFiles/p961pap.pdf (current January 2011).

Das, S., W.G. Lutters and C.B. Seaman (2007) “Understanding Documentation Value in Software Maintenance”, Proceedings of the 2007 Symposium on Computer Human Interaction for the Management of Information Technology, March 30-31, 2007, Cambridge, MA.

Davis, T., T. Thomas and A. Kazlauskas (2006) “What Were You Thinking? Empowering Tomorrow's Professionals Today”, International Journal of Pedagogies and Learning (e-Journal http://ijpl.usq.edu.au), 2(1).

de Souza, S. C., N. Anquetil and K.M. de Oliveira (2007) “Which Documentation for Software Maintenance?”, Journal of the Brazilian Computer Society, 13(2), pp. 31-44.

Dias, M. G. B., N. Anquetil and K.M. de Oliveira (2003) “Organizing the Knowledge Used in Software Maintenance”, Journal of Universal Computer Science 9(7), pp. 641-658.

Glass, R. (2003) Facts and Fallacies of Software Engineering, New York: Addison-Wesley.

Grubb, P. and A.A. Takang (2003) Software Maintenance: Concepts and Practice. World Scientific Publishing Company

Holgeid, K.K., J. Krogstie and D.I.K. Sjøberg (2000) “A Study of Development and Maintenance in Norway: Assessing the Efficiency of Information Systems Support Using Functional Maintenance”, Information and Software Technology, 42(10), pp. 687-700.

Hunt, A. and D. Thomas (2000) The Pragmatic Programmer: From Journeyman to Master. Addison-Wesley.

Kajko-Mattsson, M. (2008) “Problems in Agile Trenches”, Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, October 09-10, 2008, Kaiserslautern, Germany.

Karahasanović, A. and R.C. Thomas (2007) “Difficulties Experienced by Students in Maintaining Object-Oriented Systems: An Empirical Study”, The 9th Australasian Computing Education Conference (ACE2007), Ballarat, Victoria, Australia.

Keirnan, T., L. Anschuetz and S. Rosenbaum (2002) “Combining Usability Research with Documentation Development for Improved User Support”, Proceedings of the ACM-SIGDOC Conference, pp. 84-89.

Kernighan, B. and R. Pike (1999) The Practice of Programming. Addison-Wesley Professional.

Lehman, M. M. (1980) “Lifecycles and the Laws of Software Evolution”, Proceedings of the IEEE, Special Issue on Software Engineering, 19, pp. 1060-1076.

Lehman, M. M. (1984) “Program Evolution”, Journal of Information Processing Management, 19(1), pp. 19-36.

Lehman, M. and L.A. Belady (1985) Program Evolution – Processes of Software Change. London: Academic Press.
Lenic, M., M. Zorman, P. Povalej and P. Kokol (2004) "Alternative Measurement of Software Artifacts", Second IEEE International Conference on Computational Cybernetics, 2004, pp. 231-235.
Lientz, B.P. (1983) "Issues in Software Maintenance", Computing Surveys, 15(3), pp. 271-278.
McConnell, S. (2004) Code Complete: A Practical Handbook of Software Construction (2nd Ed.). Microsoft Press.
Naur, P. and B. Randell (1968) "Software Engineering: Report of a Conference Sponsored by the NATO Science Committee", Garmisch, Germany: Scientific Affairs Division, NATO. http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1968.PDF. (Current January, 2011)
Nosek, J.T. and P. Prashant (1990) "Software Maintenance Management: Change in the Last Decade", Journal of Software Maintenance Research and Practice, 2(3), pp. 157-174.
Osborne, W. M., E.J. Chikofsky (1990) "Fitting Pieces to the Maintenance Puzzle", IEEE Software, 7(1), pp. 11-12.
Schneidewind, N. F. (1987) "The State of Software Maintenance", IEEE Transactions on Software Engineering, 13(3), pp. 303-310.
Trienekens, J., R. Kusters et al. (2001) "Product Focused Software Process Improvement: Concepts and Experiences from Industry", Software Quality Journal 9(4), pp. 269-281.
Venable, J. (2008) "A Curriculum and Pedagogy for Teaching Problem Analysis to First Year Students", Proceedings of the AIS-SIGED IAIM 2007 Conference.
Yadin, A. and I. Lavy (2008) "Integrated Formative Assessment as a Vehicle Towards Meaningful Learning in Systems Analysis and Design Workshop", ICIS 2008 SIG-ED, Paris, France.
Yadin, A. and I. Lavy (2008) "Raising Awareness to the Constituents of Software Design – The Case of Documentation", J. Software Engineering & Applications, 2010, 3: 495-502.
Zou, Y. and K. Kontogiannis (2002) "Migration to Object Oriented Platforms: A State Transformation Approach", Proceedings of the IEEE International Conference on Software Maintenance, (ICSM'02) October 2002, Montreal, Canada, pp. 530-539.